Nereus Ecosystem Contracts Code Audit by Ambisafe Inc.

July, 2022

Oleksii Matiiasevych, Artem Martiukhin

1. **Introduction.** Nereus Finance. requested Ambisafe to perform a code audit of the contracts implementing Nereus staking, oracles and permissions. The contracts in question can be identified by the following git commit hash:

   ```
   5109349c5f9aaa5b10e7623ff4d10a9b891a71c3
   ```

   There are 9 contracts/libraries in scope.

   After the initial code audit, Nereus Finance team applied a number of updates which can be identified by the following git commit hash:

   ```
   TBD
   ```

   Additional verification was performed after that.

2. **Disclaimer.** The code audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts for any specific purpose, or their bugfree status. The code audit documentation below is for internal management discussion purposes only and should not be used or relied upon by external parties without the express written consent of Ambisafe.

3. **Executive Summary.** All the initially identified, minor and above, severity issues were **fixed** and are not present in the **final version** of the contracts. There are **no** known compiler bugs for the specified compiler version (0.6.12), that might affect the contracts' logic. There were 0 critical, 0 major, 4 minor, 3 informational and optimizational issues identified in the initial version of the contracts. The non-informational issues found in the contract were not present in the final version. They are described below for historical purposes.
   Modifications of the DegenBox mainly consist of introducing a pool indexing.

4. **Critical Bugs and Vulnerabilities.** No critical bugs or vulnerabilities were found.

5. **Initial Fixed Issues Line By Line Review.**

5.1. DegenBox, line 772. Minor, the **deposit()** function does not validate **MINIMUM_SHARE_BALANCE** condition in the case when the user specifies how many shares to mint. This allows a malicious actor to set a minimum deposit limit for a token to some high value that will restrict some users from depositing.

5.2. DegenBox, line 902. Minor, the **transferMultiple()** function first adds the balance to the receiver then subtracts from the sender. In case the sender and receiver are the same address then a malicious actor could emit fake **LogTransfer** events not having any balance at all.

5.3. DegenBox, line 929. Minor, the **flashLoan()** function does not verify if the token was already deposited or not, allowing a malicious actor to increase the **totals[token].elastic** value while keeping the **totals[token].base** equaling zero.

5.4. DegenBox, line 964. Minor, the **batchFlashLoan()** function does not verify if the token was already deposited or not, allowing a malicious actor to increase the **totals[token].elastic** value while keeping the **totals[token].base** equaling zero.

5.5. PermissionManager, line 22. Note, the **permit()** function does not emit any events.

5.6. PermissionManager, line 37. Note, the **revoke()** function does not emit any events.

5.7. WhitelistManager, line 10. Note, the **setCheckStatus()** function does not emit any events.

6. **Remaining Issues Line By Line Review.**

[illegible]

Oleksii Matiiasevych